Theoretical Computer Science 110 (1993) 53–78
Elsevier

On the limit set of some universal cellular automata*

Eric Goles, Alejandro Maass and Servet Martinez

Universidad de Chile, Facultad de Ciencias Físicas Matemáticas, Departamento de Ingeniería Matemática, Casilla 170-3 correo 3, Santiago, Chile

Communicated by D. Perrin
Received February 1991
Revised September 1991

Abstract

Goles, E., A. Maass and S. Martinez, On the limit set of some universal cellular automata, Theoretical Computer Science 110 (1993) 53–78.

In this paper we construct a simulation of a Turing machine by cellular automata based on the equivalence between programmable machines and Turing machines. For this class of cellular automata we prove that the associated limit language is regular. We also introduce algebraic characteristics which reduce the study of the complexity of the limit language to the analysis of a class of one-symbol languages.

1. Introduction

Let (S, r, f, F) be a linear cellular automaton (CA): S is the finite alphabet (or state set), r is the neighborhood radius, $f: S^{2r+1} \to S$ is the local rule and $F: S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ is the global transition function associated with f, i.e. $(F(c))_i = f(c_{i-r}, \ldots, c_i, \ldots, c_{i+r})$ for all $i \in \mathbb{Z}$ and $c \in S^{\mathbb{Z}}$.

To study some aspects of the asymptotic dynamics of CAs, Wolfram [18] introduced the notion of limit set associated with a CA, $\Lambda(F) = \bigcap_{t \geq 0} F^t(S^{\mathbb{Z}})$, which represents the configurations with infinite preimages. Let us define the language associated with the limit set, $\mathcal{L}(\Lambda(F))$, as the collection of all finite blocks contained in the configurations of $\Lambda(F)$. It is well known that, for finite time, i.e. $\bigcap_{t=0}^{q} F^t(S^{\mathbb{Z}}) = F^q(S^{\mathbb{Z}})$, the associated language $\mathcal{L}(F^q(S^{\mathbb{Z}}))$ is regular [16]. Wolfram

Correspondence to: E. Goles, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, Departamento de Ingeniería Matemática, Casilla 170-3 correo 3, Santiago, Chile.

* This work has been done under grants FONDECYT 1211-91, 0040-90, 1208–91.

[18] set the problem whether for any CA, the limit language $\mathscr{L}(\Lambda(F))$ is regular. This question was answered by Hurd in [8], where some examples are given of CAs which limit sets have nonregular context-free or non-context-free context-sensitive, or non-recursive enumerable limit languages. Another result was furnished by Cullik [3], who showed that the limit language of some CA that simulates the dynamics of any CA is nonrecursive enumerable.

For the most complex limit languages obtained by previous authors, they had to simulate universal Turing machines by CAs. Based on this result, one could think that the computational complexity is always translated to the limit language. In this paper, we prove that it is not true, i.e. there exist CAs that simulate any Turing machine (TM) whose limit languages are regulars.

In Section 2, we propose a morphism between programmable machines (PM) and CAs. By using the Minsky's equivalence between TM and PM, we construct a CA that simulates any Turing machine. This simulation is used to prove the results on the limit language in Section 3.

In Section 3, we study the limit language complexity associated with the CA developed in Section 2. Here, we introduce the concept of perturbation for a PM, that is a modification of the original PM keeping the finite-time behaviour, but introducing inverse dynamics for some instructions in the PM program. Using perturbed PM, we prove that it is possible to simulate a universal TM by a CA whose limit language is regular. From this result we can see that there is no direct relation between computational complexity of a CA and the limit language complexity.

In Section 4, we study the limit language for nonperturbed PM. We prove that the CA which simulates a universal Turing machine has nonrecursive enumerable limit language. For more general PM we establish that the regularity of the limit language depends on the finiteness of some algebraic characteristic of the PM. This finiteness is reached by the perturbed PM. Also, we prove that CAs simulating a finite automaton have a finite characteristic, thus a regular limit language.

Finally, in Section 5, by considering general perturbed PM, we recover the same classes of limit languages as exhibited in [8]. Moreover, we introduce a finite algorithm to study the limit language complexity for the CAs associated with this class of programmable machines, which depends only on the associated connection graph.

2. Universal cellular automaton

We will give a simulation of a Turing machine by a cellular automaton. This construction will allow us to study the limit complexity in Section 3. For the construction, we use the classical equivalence between Turing machines and programmable machines established in [15].

Throughout this paper we define a Turing machine, M_T, as the octuple $M_T=(Q, A, \alpha, \beta, \gamma, q_s, q_h, b)$. Q is a finite set of inner states, $q_s, q_h \in Q$ are an initial and a final state, respectively. The set $A=I \cup W$ is a finite alphabet, where we distinguish the input

alphabet I and the work alphabet W, $b \in W$ is the empty letter. Finally, α, β, γ are the transition functions associated with internal states, symbols and head translations, respectively:

$$\alpha : (Q \setminus \{q_h\}) \times A \to \mathscr{P}(Q),$$

$$\beta : (Q \setminus \{q_h\}) \times A \to \mathscr{P}(W),$$

$$\gamma : (Q \setminus \{q_h\}) \times A \to \{-1, 1\}.$$

A programmable machine, M_p, is the triple $M_p = (\mathscr{R}, \mathscr{I}, \mathscr{P})$. Here $\mathscr{R} = \{R_1, \ldots, R_N\}$ is a finite collection of registers storing nonnegative integers, the set $\mathscr{I} = \{I_1, \ldots, I_s\}$ corresponds to the permissible instructions over the registers and $\mathscr{P} = (p_1, p_2, \ldots, p_{k_0}) = (I_{j_1}, \ldots, I_{jk_0})$ is a program with k_0 instructions belonging to the set $\mathscr{I}$.

The dynamics of a programmable machine consist in updating the registers R_j, $j = 1, \ldots, N$, according to the program from an initial condition at the registers.

The set of permissible instructions can be summarized as follows:

R_j^+ : Add one in the register R_j, and go to the next instruction.

$R_j^-(n)$: Subtract one in the register R_j if it is not empty, and go to the next instruction. If R_j is empty, go to the nth instruction of $\mathscr{P}$.

$Go(n)$: Go to the nth instruction of $\mathscr{P}$.

H : Halt instruction.

We identify a programmable machine, M_p, with its connection graph $G(M_p) = (\mathscr{N}(M_p), \mathscr{E}(M_p))$. The set of nodes is $\mathscr{N}(M_p) = \{n_1, \ldots, n_{k_0}\}$, where $n_i = (i, p_i)$ for $i = 1, \ldots, k_0$. An edge $(n_i, n_j) \in \mathscr{E}(M_p)$ if the instruction j is attained in one step from instruction i.

Often, the nodes $n_i = (i, p_i)$ are referred to indistinctly by i, which codes the instruction number, or by p_i, referring to the type of instruction.

We support our simulation in the following equivalence theorem between Turing machines and programmable machines.

Theorem 2.1 (Minsky, [15]). (1) *Given a Turing machine M_T, there exists a programmable machine M_p which simulates it with two registers $\mathscr{R} = \{A, B\}$.*

(2) *For any programmable machine M_p, there exists a Turing machine M_T which simulates it.*

It is important to point out that the equivalence is established by using simple connections of a finite number of two kinds of blocks: the increment block, IB, and the decrement block, DB, as is shown in Fig. 1. This remark will be cross-shaped to study the complexity of the limit set.

2.1. Codification of a programmable machine by a cellular automaton

From Theorem 2.1, we see that a programmable machine with two registers is enough to simulate a universal Turing machine. This result leads us to construct a CA which simulates any programmable machine with two registers.

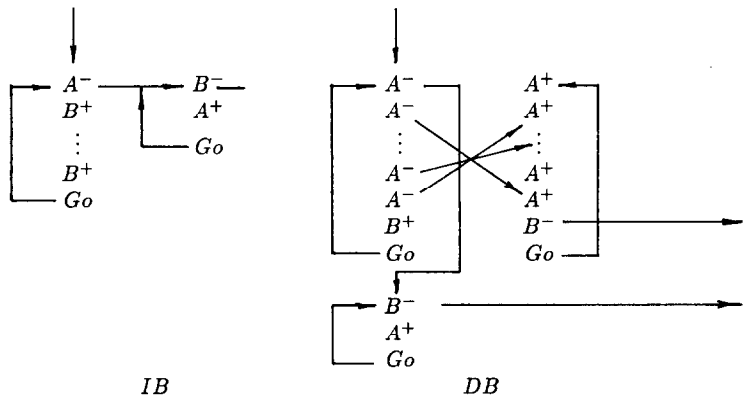


Fig. 1. Classical blocks of instructions which compose the equivalent programmable machine given in Theorem 2.1. In block IB the size of the blocks of states B^+ vary from 2 to 7. Similarly, in block DB the size of the blocks of states A^- and A^+ are the same and vary from 2 to 7.

Let M_p be a programmable machine with two registers. From an initial condition at the registers A and B, the programmable machine runs accordingly with the program $\mathscr{P}$. At each step, only one instruction of $\mathscr{P}$ will be active which updates the corresponding register. In order to achieve this process, the corresponding register has to be reported, and once the operation is done, the central unit of the program has to change to the next instruction, and so on.

In our simulation, the cellular space is divided into three sectors: (a) register A, (b) register B, (c) central unit of instructions (CUI). The communications are carried by states, called vehicles.

The states of the simulation are the following:

$Q_1 = \{0\}$: quiescent state

$Q_2 = \{F_A, F_B\}$: delimiters for the registers

$Q_3 = \{A^+, A^-, B^+, B^-, G\}$: states associated with each type of instruction (A^+, $A^-(n)$, B^+, $B^-(n)$, $Go(n)$, respectively)

$Q_4 = \{r, l\}$: states associated with the end of the execution of each instruction (r with register A and l with register B)

$Q_5 = \{n_1, \ldots, n_{k_0}\}$: states associated with the nodes of the program $\mathscr{P}$ of M_p.

Then, $S = \bigcup_{j=1}^{5} Q_j$ and $|S| = k_0 + 10$, where k_0 is the size of $\mathscr{P}$.

For the simulation, the cellular space distribution is shown in Fig. 3. There, the contents $|A|$ and $|B|$ of the registers, are coded by a sequence of zeros of size $|A|$ and $|B|$, between the corresponding markers and the central unit. The markers F_A and F_B are repeated infinitely to the left and right of the coded contents $|A|$ and $|B|$.

All the states in $Q_3 \cup Q_4$ act like vehicles. We will call right vehicles the states B^+, B^-, G, r and left vehicles the states A^+, A^-, l. A right vehicle will be denoted by

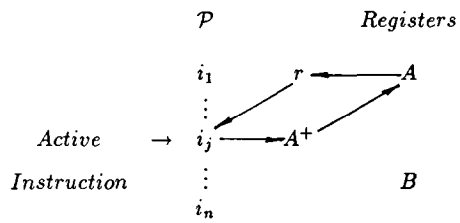


Fig. 2. The active node $n_j=(i_j, A^+)$ sends a vehicle A^+ to the corresponding register. When the instruction has been carried out, it sends, to the central unit, a new vehicle r to make active the next instruction. The case $n_j=(i_j, A^-(m))$ is analogous except when register A is empty. In this case the instruction m is immediately activated.

$$\cdots F_A\ F_A\ 0 \cdots 0\ i\ n_j\ d\ 0 \cdots 0\ F_B\ F_B \cdots$$
Register A Central unit Register B

Fig. 3. Cellular space distribution; $i\in\{r, A^+, A^-, 0\}$, $d\in\{l, B^+, B^-, G, 0\}$.

R and a left one by L. This name is according to the type of action that they realize (see Table 1).

Definition 2.2. (1) A configuration $c\in S^{\mathbb{Z}}$ is a "simulating configuration" (SC) if

(a) it takes the form of Fig. 2, and

(b) the central unit is $V_i n_j V_d$, where $V_i=0$ or the vehicle associated with $n_j=(j, p_j)$ if p_j is A^+ or A^-, and V_d is analogously defined as V_i, but in register B. If $p_j=G$ then $V_d=G$.

(2) A configuration $c_0\in S^{\mathbb{Z}}$ is an "initial SC" (ISC) if

(a) c_0 is a SC and

(b) the central unit is $V_i n_1 V_d$.

From an ISC, c_0, the automaton develops a sequence $(c_t)_{t\geq 1}$ of SCs such that if c_t and c_{t+1} have central units (V_i, n_{j_t}, V_d) and $(V_i, n_{j_{t+1}}, V_d)$, respectively, then

(1) if $n_{j_t}=(j_t, A^+)$, the vehicle A^+ runs up to F_A, erasing it, and generating a return vehicle, the state r, which runs up to the central unit activating the next central unit $(V_i, n_{j_{t+1}}, V_d)$ of c_{t+1};

(2) if $n_{j_t}=(j_t, A^-(n))$ it is analogous to (1), but adding one delimiter F_A, if the subtraction can be done. If not, the next state is automatically activated;

(3) if $n_{j_t}=(j_t, G)$, this vehicles runs up to delimiter F_B and returns the state l without change at the register.

The cases of B^+ and B^- are analogous to (1) and (2), respectively.

Recall that the sequence $(c_t)_{t\geq 1}$ does not appear in real time except when the central unit changes. In fact, time depends on the size of the registers.

This CA is characterized by its local rule, f, given in Table 1, which has a neighborhood radius $r=3$. The choice of $r=3$ is made in order to solve the case of an empty register with an instruction A^-. The table is divided according to the different kinds of

Table 1. Local rule for the CA constructed Section 2. "·" represents an arbitrary state in S, R represents a right vehicle and L a left vehicle

Interactions	c_{-3}	c_{-2}	c_{-1}	c_0	c_1	c_2	c_3	$f(c_{-3},c_{-2},c_{-1},c_0,c_1,c_2,c_3)$
(1) Vehicles	·	·	R	x	y	·	·	$R(x\neq L, y\neq L)$
	·	·	y	x	L	·	·	$L(x\neq R, y\neq R)$
(2) Vehicles and markers	·	·	B^+	F_B	F_B	·	·	1
	·	·	x	B^+	F_B	·	·	$0\ (x\neq R)$
(in A is analogous)	·	·	R	B^+	F_B	·	·	R
	·	·	x	r	F_B	·	·	$0\ (x\neq R)$
	·	·	R	r	F_B	·	·	R
	·	0	B^-	F_B	F_B	·	·	F_B
	·	·	0	B^-	F_B	·	·	F_B
	·	·	x	0	B^-	F_B	·	$l\ (x\neq R)$
	·	·	R	0	B^-	F_B	·	0
	·	·	G	F_B	F_B	·	·	F_B
	·	·	x	G	F_B	·	·	$l\ (x\neq R)$
	·	·	R	G	F_B	·	·	0
(3) Special rules for markers	·	·	x	F_A	·	·	·	$0\ (x\neq F_A)$
	·	·	·	F_B	x	·	·	$0\ (x\neq F_B)$
	·	·	0	F_B	F_B	y	·	$F_B\ (y\neq F_A)$
	·	y	F_A	F_A	0	·	·	$F_A\ (y\neq F_B)$
	·	·	·	x	F_A	·	·	$0\ (x\neq F_A)$
	·	·	F_B	x	·	·	·	$0\ (x\neq F_B)$
(4) Interactions with instructions	·	·	·	n_j	·	·	·	n_j if $n_j=(j,H)$
	·	·	x	n_j	y	·	·	$n_j\ (xy\neq 0l)$
	·	·	0	n_j	l	·	·	n_{j+1} if $n_j\neq(j, Go(n_k))$
Valid for instructions associated with register B	·	·	·	0	n_j	l	·	n_k if $n_j=(j, Go(n_k))$ 0 if n_{j+1} associated with B $A^+\vee A^-$ if n_{j+1} associated with A
	·	0	n_j	l	·	·	·	0 if n_{j+1} associated with A $B^+, B^-\vee G$ if n_{j+1} associated with B
$n_j=(j, B^-(n_k))$	·	0	n_j	B^-	F_B	·	·	0 if n_k associated with A $B^+, B^-\vee G$ if n_k associated with B
	·	·	·	n_j	B^-	F_B	·	n_k
	·	·	·	0	n_j	B^-	F_B	0 if n_k associated with B $A^+\vee A^-$ if n_k associated with B
(5)	Other cases							0

interactions between the states. For simplicity, we just describe the rules for one register. For the other register, the rules are analogous. Observe that the rules are not disjoint from one item to another; however, the rules will be understood according to their actions. The transitions that can be done with more than one rule take the rule

t	Register A					CUI			Register B						
1	$\cdots$	F_A	F_A	0	0	0	A^-	n_1	0	F_B	F_B	F_B	F_B	F_B	F_B	$\cdots$
2	$\cdots$	F_A	F_A	0	0	A^-	0	n_1	0	F_B	F_B	F_B	F_B	F_B	F_B	$\cdots$
3	$\cdots$	F_A	F_A	0	A^-	0	0	n_1	0	F_B	F_B	F_B	F_B	F_B	F_B	$\cdots$
4	$\cdots$	F_A	F_A	A^-	0	0	0	n_1	0	F_B	F_B	F_B	F_B	F_B	F_B	$\cdots$
5	$\cdots$	F_A	F_A	F_A	r	0	0	n_1	0	F_B	F_B	F_B	F_B	F_B	F_B	$\cdots$
6	$\cdots$	F_A	F_A	F_A	0	r	0	n_1	0	F_B	F_B	F_B	F_B	F_B	F_B	$\cdots$
7	$\cdots$	F_A	F_A	F_A	0	0	r	n_1	0	F_B	F_B	F_B	F_B	F_B	F_B	$\cdots$
8	$\cdots$	F_A	F_A	F_A	0	0	0	n_2	B^+	F_B	F_B	F_B	F_B	F_B	F_B	$\cdots$
9	$\cdots$	F_A	F_A	F_A	0	0	0	n_2	0	l	F_B	F_B	F_B	F_B	F_B	$\cdots$
10	$\cdots$	F_A	F_A	F_A	0	0	0	n_2	l	0	0	F_B	F_B	F_B	F_B	$\cdots$
11	$\cdots$	F_A	F_A	F_A	0	0	0	n_3	B^+	0	F_B	F_B	F_B	F_B	F_B	$\cdots$
12	$\cdots$	F_A	F_A	F_A	0	0	0	n_3	0	B^+	F_B	F_B	F_B	F_B	F_B	$\cdots$
13	$\cdots$	F_A	F_A	F_A	0	0	0	n_3	0	0	l	F_B	F_B	F_B	F_B	$\cdots$
14	$\cdots$	F_A	F_A	F_A	0	0	0	n_3	0	l	0	F_B	F_B	F_B	F_B	$\cdots$
15	$\cdots$	F_A	F_A	F_A	0	0	0	n_3	l	0	o	F_B	F_B	F_B	F_B	$\cdots$
16	$\cdots$	F_A	F_A	F_A	0	0	0	n_4	G	0	0	F_B	F_B	F_B	F_B	$\cdots$
17	$\cdots$	F_A	F_A	F_A	0	0	0	n_4	0	G	0	F_B	F_B	F_B	F_B	$\cdots$
18	$\cdots$	F_A	F_A	F_A	0	0	0	n_4	0	0	G	F_B	F_B	F_B	F_B	$\cdots$
19	$\cdots$	F_A	F_A	F_A	0	0	0	n_4	0	0	l	F_B	F_B	F_B	F_B	$\cdots$
20	$\cdots$	F_A	F_A	F_A	0	0	0	n_4	0	l	0	F_B	F_B	F_B	F_B	$\cdots$
21	$\cdots$	F_A	F_A	F_A	0	0	0	n_4	l	0	0	F_B	F_B	F_B	F_B	$\cdots$
t_0	$\cdots$	F_A	F_A	F_A	0	0	A^-	n_1	0	0	0	F_B	F_B	F_B	F_B	$\cdots$
$3t_0$	$\cdots$	F_A	F_A	F_A	F_A	F_A	A^-	n_1	0	0	0	0	0	0	0	F_BF_B
$3t_0+j, j\geqslant 1$	$\cdots$	F_A	F_A	F_A	F_A	F_A	0	H	0	0	0	0	0	0	0	F_BF_B
Fixed point																

Fig. 4. The dynamics of the PM of Example 2.4, beginning with contents $|A|=3$ and $|B|=0$ at instruction n_1. When the contents at the registers are $|A|=0, |B|=6$, the halt state is reached in 66 iterations.

that is made explicit for the case. An example of the automaton dynamics is given in Fig. 4.

It is important to note that the simulation depends on the number of instructions k_0 of the programmable machine. The set of states S has cardinality $|S|=k_0+10$. However, this point is not relevant to calculate the limit set of this automaton.

Now, we will make explicit the sense of our simulation. For this purpose we introduce the following definition.

Definition 2.3. Let $M_p=\{\{A, B\}, \mathscr{P}\}$ be a programmable machine and $F(M_p)$ be the CA defined by Table 1. Let c_0 be an ISC for M_p and $(c_t)_{t\geqslant 1}$ be the SC generated by $F(M_p)$ from the initial configuration c_0. On the other hand, let $(A_t)_{t\geqslant 0}, (B_t)_{t\geqslant 0}$ be the sequences of integers in the registers A and B generated by M_p from the initial values A_0, B_0.

Denote by c_t^A and c_t^B the contents of the registers A and B in the SC, $\{c_t\}_{t\geqslant 1}$. We say that $F(M_p)$ simulates M_p if

$$c_0^A=A_0 \text{ and } c_0^B=B_0 \Rightarrow c_t^A=A_t \text{ and } c_t^B=B_t, \quad \forall t\geqslant 1.$$

Let us consider a Turing machine M_T and the programmable machine M_p, associated with M_T, by Theorem 2.1. For M_p, we can determine a CA, $F(M_p)=F$, which simulates it, in the sense of Definition 2.2. Then, we say that F simulates M_T, in the same sense. Observe that the programmable machine considered in this simulation is a simple connection of blocks DB and IB (see Fig. 1).

We will say that a CA, F, is universal if it simulates a universal Turing machine. Since a programmable machine may simulate a universal Turing machine, previous cellular automaton is computational universal.

Example 2.4. Let $M_p=(\{A, B\}, \{A^+, A^-(n), B^+, B^-(n), Go(n)\}, \mathscr{P})$, with $\mathscr{P}=(A^-(5), B^+, B^+, Go(1), H)$. This programmable machine multiplies by two the content of register A, and stores the result in register B. The simulated CA is given in Fig. 4.

3. Limit complexity for the universal CA

Let M_T be a Turing machine and F a one-dimensional CA simulating M_T via its associated programmable machine. For F, define the associated limit language as $\mathscr{L}_F=\mathscr{L}(\Lambda(F))=\mathscr{L}(\bigcap_{i\geq 0} F^i(S^{\mathbb{Z}}))$, which represents all the finite blocks that appear in configurations of the limit set $\Lambda(F)$.

In this section, we characterize the limit languages for some kind of simulations, $\tilde{F}$, based on the scheme given in Section 2. We prove that the limit languages, $\mathscr{L}_{\tilde{F}}$, are regulars for any Turing machine M_T. In this case, $\tilde{F}$ is a regular codification of M_T. First, let us introduce some definitions.

Definition 3.1. Let $M_p=(\mathscr{R}, \mathscr{I}, \mathscr{P})$ be a programmable machine, with $\mathscr{R}=\{R_1, \ldots, R_N\}$. Consider the set $\tilde{\mathscr{P}}=\{n_i\in\mathscr{N}(M_p) | n_i=(i, R_k^-(j))$ for some $j\in\{1, \ldots, k_0\}$ and some $k\in\{1, \ldots, N\}\}$.

(1) For each $n_i\in\tilde{\mathscr{P}}$ define the associated "subblock" or "external loop" as the following pair of instructions:

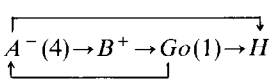
$$n_{i^1}=(i^1, R_k^-(i)),$$

$$n_{i^2}=(i^2, Go(i^1)),$$

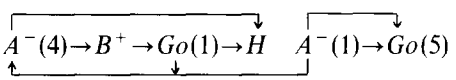
where the sets $\{i^1 | n_i\in\tilde{\mathscr{P}}\}$ and $\{i^2 | n_i\in\tilde{\mathscr{P}}\}$ are disjoints, ordered from k_0+1 to the finish and such that $i^2=i^1+1$.

(2) The perturbed programmable machine, M'_p, associated with M_p is $M'_p=(\mathscr{R}, \mathscr{I}, \mathscr{P}')$, where $\mathscr{P}'$ is the program constructed with $\mathscr{P}$ and the subblocks associated with $\tilde{\mathscr{P}}$.

Example 3.2. Consider the programmable machine M_p defined through the graph:



Then M'_p is given by



We can extend this concept for a Turing machine, taking the perturbed programmable machine associated with it, by Theorem 2.1.

Definition 3.3. Take the Turing machine M_T. Let M_p be the programmable machine associated with it (by Theorem 2.1) and M'_p its perturbation. Consider the cellular automata $F, \tilde{F}$ defined in Fig. 2.4 that simulate them. Then

(1) F will be called the normal simulator of M_T (NS),

(2) $\tilde{F}$ will be called the perturbated simulator of M_T (PS).

This concept allows us to study the inverse dynamics associated with the CA and regularize the limit languange.

Theorem 3.4. *Let $\tilde{F}$ be the associated PS of a Turing machine M_T. Then $\mathscr{L}_{\tilde{F}}$ is regular.*

A detailed proof of this theorem is difficult; so, we divide it in two parts. The complexity of the limit language depends strongly on the type of interactions between states belonging to the same configuration. The states of each configuration can be the markers F_A or F_B, vehicles, states associated with the instructions and the quiescent state 0. So, we must take into account the relations between such states and partition the set of configurations according to the type of interactions that they can support.

In the first part of the theorem, we consider the set of configurations that have interactions between at most two types of states in a background of zeros. So, we consider interactions of a state with itself, interactions between markers and vehicles, between markers and instructions and between vehicles and instructions. We prove that the associated components of the limit language for this class of configurations are regular (Proposition 3.9). These configurations are called nondynamical configurations.

In the second part of the theorem, we study the limit set components associated with configurations which have interactions between markers, instructions and vehicles in a background of zeros. These kinds of configurations introduce the problem of the inverse dynamics of the simulated programmable machine. These configurations are called dynamical configurations. In this case, we also prove that the components in the limit language are regular (Proposition 3.13). In Proposition 3.13 the perturbed cellular automaton $\tilde{F}$ associated with F is cross-shaped.

To prove the first part, we need several lemmas, with easy proofs, that characterize $\tilde{F}$.

First, let us show that all right vehicles must appear at the left side of a left vehicle in the limit set

Lemma 3.5. *Let* $L \in \{A^+, A^-, l\}$ *and* $R \in \{B^+, B^-, G, r\}$; *then*

$$\{L0^*R\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$$

Proof. It follows directly from the definition of the local rules. □

Now, we prove some technical results which allow us to describe the configurations of the limit language containing markers.

Lemma 3.6. *If* $X \in (Q_1 \cup Q_3 \cup Q_4 \cup Q_5)^*$, *then*

(i) $\{F_B X F_A\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$.

If $X \in \{B^+, B^-, G, r\} \cup Q_5$ *and* $Y \in S^*$, *then*

(ii) $\{F_B Y X\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$.

If $X \in \{A^+, A^-, l\} \cup Q_5$ *and* $Y \in S^*$, *then*

(iii) $\{X Y F_A\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$.

If $X \in (Q_1 \cup Q_3 \cup Q_4 \cup Q_5)^*$ *and* $Y \in (Q_1 \cup Q_3 \cup Q_4 \cup Q_5)$, *then*

(iv) $\{F_B Y X F_B\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$,

(v) $\{F_A Y X F_A\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$.

If $X \in \{B^+, B^-, G\}$, *then*

(vi) $\{F_A 0^* X\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$.

If $X \in \{A^+, A^-\}$, *then*

(vii) $\{X 0^* F_B\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$.

If $R \in \{B^+, B^-, G, r\}$ *and* $L \in \{A^+, A^-, l\}$, *then*

(viii) $\{F_A 0^* r 0^* R\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$,

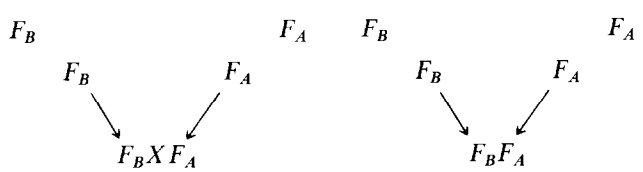
(ix) $\{L0^* l 0^* F_B\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$.

If $X \in \{A^+, A^-, l, B^+, B^-, G, r\}$ *then*:

(x) $\{F_A (X \vee 0)^* X (X \vee 0)^* F_B\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$.

Proof. We just prove (i) and (ii); the other cases are analogous.

(i) From rules (2) and (3) in Table 1, it follows directly to get that the markers F_B generate their sequence moving to the right in the preimages, and the markers F_A generate a similar sequence to the left. But the words $F_B X F_A$ and $F_B F_A$ do not have preimages by $\tilde{F}$; then these kinds of words do not belong to the limit language:



(ii) As in the previous proof, every marker F_B must have another marker F_B at the right in the preimage. If $X \in Q_5$, the reverse dynamics form a vertical block of

instructions in the preimage. This and the first point show (ii):

$$\begin{array}{cccc} F_B & Y & X & \\ & & \vdots & \in Q_5 \\ & & F_B & \notin Q_5 \end{array}$$

If X is a right vehicle, it follows directly from (i), Lemma 3.1 and the first part that we just have to analyse the case of the word $F_B 0^* R$. But the markers F_B do not produce any right vehicle; then this word is not in the limit language:

$$\begin{array}{ccccccccc} F_B & 0 & 0 & 0 & 0 & 0 & R & & \\ & F_B & & & & R & & & \\ & & F_B & 0 & R & & & & \\ & & & F_B & & & & & \square \end{array}$$

Now, in the next lemma we establish the relations between states $n_k \in Q_5$ and vehicles. For this purpose, we partition Q_5 as follows:

$$Q_5 = I_A^+ \cup I_A^- \cup I_B^+ \cup I_B^- \cup I_G \cup I_H,$$

where

$$I_e^+ = \{n_k = (k, p_k) \in Q_5 \mid p_k = e^+\}, \quad e = A, B,$$

$$I_e^- = \{n_k = (k, p_k) \in Q_5 \mid p_k = e^-(n), n \in Q_5\}, \quad e = A, B,$$

$$I_G = \{n_k = (k, Go(j)) \in Q_5 \mid j \in \{1, \ldots, k_0\}\},$$

$$I_H = \{n_k = (k, H) \in Q_5\}.$$

We associate with each $n_k \in Q_5$ the left vehicle (LV) and right vehicle (RV) by

$$V^{\mathrm{L}}(n_k) = \mathrm{LV} \quad \text{associated with } n_k = \begin{cases} A^+ & n_k \in I_A^+, \\ A^- & n_k \in I_A^-, \\ 0 & \sim, \end{cases}$$

$$V^{\mathrm{R}}(n_k) = \mathrm{RV} \quad \text{associated with } n_k = \begin{cases} B^+ & n_k \in I_B^+, \\ B^- & n_k \in I_B^-, \\ G & n_k \in I_G, \\ 0 & \sim. \end{cases}$$

It follows directly that $V^L(n_k) = 0$ or $V^{\mathrm{R}}(n_k) = 0$.

Lemma 3.7. *Let* $n_k \in Q_5$ *and* $R \in \{B^+, B^-, G, r\}$, $L \in \{A^+, A^-, l\}$; *then*:

(i) $\{n_k 0^* r\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$,

(ii) $\{l 0^* n_k\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$,

(iii) $\{L 0^* V^{\mathrm{L}}(n_k) 0^* n_k 0^* V^{\mathrm{R}}(n_k)\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$,

(iv) $\{V^{\mathrm{L}}(n_k) 0^* n_k 0^* V^{\mathrm{R}}(n_k) 0^* R\} \cap \mathscr{L}_{\tilde{F}} = \emptyset$.

Proof. We just prove (iii). Without loss of generality, suppose $n_k=(k, B^+)$. Then $V^L(n_k)=0$, $V^R(n_k)=B^+$. Assume that there exists $n_j\in Q_5$ such that $(n_j, n_k)\in\mathscr{E}(M_p)$. From rules (4) in Table 1, the states of instructions form a vertical block in the preimages, and the active instruction is deactivated only when the instruction has a state l at the right, if the instruction is associated with register B, or a state r at the left in the other cases. The situation is the following:

$$\begin{array}{cccccc} L & 0^* & 0 & n_k & B^+ \\ & & i & n_j & j \end{array}$$

There are two cases:

(1) If $i=r$, $j=0$, we conclude the proof by Lemma 3.5.

(2) If $i=0$, $j=l$, we conclude the proof because n_j is associated with B. □

From previous lemmas, the general structure of the limit words is the following:

$$(R\vee 0)^* w_1 F_A^* w_2 F_B^* w_3 (L\vee 0)^*,$$

where $w_1, w_2, w_3\in S^*$ are determined by the existence of markers and states of instruction in the limit word.

Definition 3.8. Let M_p be a Turing machine and $G(M_p)$ the graph of its associated programmable machine. Consider a node $n_i=(i, p_i)\in\mathscr{N}(M_p)$.

(1) The set of infinite preimages associated with n_i is defined as follows:

$$\mathscr{A}(n_i)=\{a\in(\{n_i\}\times(\mathscr{N}(M_p))^{\mathbb{N}})\mid(a_{j+1}, a_j)\in\mathscr{E}(M_p) \text{ for } j\geqslant 1 \text{ and } (a_1, n_i)\in\mathscr{E}(M_p)\}.$$

(2) The set of predecessors associated with n_i is defined as

$$\mathscr{A}_1(n_i)=\{n_j\in\mathscr{N}(M_p)\mid(n_j, n_i)\in\mathscr{E}(M_p)\}.$$

Now, we can prove the first part of Theorem 3.4.

Proposition 3.9. *Let M_T be a Turing machine and $\tilde{F}$ the PS which simulates it. Then, the limit language component associated with nondynamical configurations is regular.*

Proof. We have to go over all the words that belong to configurations which have infinite preimages by $\tilde{F}$. In this proposition, we just consider words containing at most two types of states interacting in a background of zeros. We will describe, with the usual notation of regular languages, each component of the limit language. Denote by $\Lambda(\tilde{F})_i$ these components and, in the case $\Lambda(\tilde{F})_i$ is partitioned, we denote its components by $\Lambda(\tilde{F})_{i,j}$.

Recall that $R\in\{B^+, B^-, G, r\}$ and $L\in\{A^+, A^-, l\}$.

(1) *Vehicles interaction.* From Lemma 3.5 and rules (1) in Table 1, the only associated component in the limit language is

$$\Lambda(\tilde{F})_1=\{(R\vee 0)^*(L\vee 0)^*\},$$

which is a regular language.

(2) *Interaction between vehicles and markers.* From previous lemmas, the general structures of this type of words are

$$\{w_1 F_A F_A^* w_2 (L \vee 0)^*\}, \tag{3.1}$$

$$\{(R \vee 0)^* w_1 F_B F_B^* w_2\}, \tag{3.2}$$

$$\{w_1 F_A F_A^* w_2 F_B^* F_B w_3\}. \tag{3.3}$$

From Lemma 3.6(viii), (iii) and rules (2) in Table 1, in (3.1) $w_1 = ((R \vee 0)^* 0) \vee \xi$ and $w_2 = 0^*(r \vee \xi)$, where ξ is the empty word. Analogously for (3.2). In (3.3), Lemma 3.6(x) (iii) imply that $w_1 = ((R \vee 0)^* 0) \vee \xi$, $w_2 = 0^*$, $w_3 = (0(L \vee 0)^*) \vee \xi$. So, the associated components of the limit language are

$$\Lambda(\tilde{F})_{21} = \{(((R \vee 0)^* 0) \vee \xi) F_A F_A^* 0^* (r \vee \xi)(L \vee 0)^*\},$$

$$\Lambda(\tilde{F})_{22} = \{(R \vee 0)^* (l \vee \xi) 0^* F_B^* F_B ((0(L \vee 0)^*) \vee \xi)\},$$

$$\Lambda(\tilde{F})_{23} = \{(((R \vee 0)^* 0) \vee \xi) F_A F_A^* 0^* F_B F_B^* ((0(L \vee 0)^*) \vee \xi)\}.$$

Then, $\Lambda(\tilde{F})_2 = \bigcup_{i=1}^{3} \Lambda(\tilde{F})_{2i}$ is a finite union of regular components. Note that in this case a simple interaction of markers was considered.

(3) *Interaction between instructions and vehicles.* We study the components of the limit language that contain only the vehicles R or L, and nodes $n_k \in Q_5$. We divide the study according to the number of states $n_k \in Q_5$ that are present.

(3.1) *One instruction.* From Lemmas 3.5 and 3.6, the structure of the words with one instruction is the following:

$$(R \vee 0)^* w_1 n_k w_2 (L \vee 0)^*, \quad n_k \in Q_5, \ w_1, w_2 \in \{0, R, L\}^*.$$

Since $n_k \in Q_5$ are fixed in the absence of vehicles, a first component is

$$\Lambda(\tilde{F})_{31}(n_k) = \{(R \vee 0)^* n_k (L \vee 0)^*\}, \qquad \Lambda(\tilde{F})_{31} = \bigcup_{n_k \in Q_5} \Lambda(\tilde{F})_{31}(n_k).$$

When $n_k \in Q_5$ and $\mathscr{A}_1(n_k) \neq \emptyset$, Lemma 3.7 gives the following limit component:

$$\Lambda(\tilde{F})_{32}(n_k) = \{(R \vee 0)^* (V^{\mathrm{L}}(n_k) \vee \xi) 0^* n_k 0^* (V^{\mathrm{R}}(n_k) \vee \xi)(L \vee 0)^*\}$$

and

$$\Lambda(\tilde{F})_{32} = \bigcup_{\substack{n_k \in Q_5 \\ \mathscr{A}_1(n_k) \neq \emptyset}} \Lambda(\tilde{F})_{32}(n_k).$$

(3.2) *Two instructions.* Let $n_k, n_l \in Q_5$. By previous lemmas, the general structure of the limit words is the following:

$$(R \vee 0)^* w_1 n_k w_2 n_l w_3 (L \vee 0)^*,$$

where w_1 has at most a vehicle $V^{\mathrm{L}}(n_k)$, w_2 has at most two vehicles, $V^{\mathrm{R}}(n_k)$ and $V^{\mathrm{L}}(n_l)$, and w_3 have at most a vehicle $V^{\mathrm{R}}(n_l)$.

As in (3.1), a direct component is the following:

$$\Lambda(\tilde{F})_{33}(n_k, n_l) = \{(R \vee 0)^* n_k 0^* n_l (L \vee 0)^*\}, \quad \Lambda(\tilde{F})_{33} = \bigcup_{(n_k, n_l) \in Q_5^2} \Lambda_{33}(\tilde{F})(n_k, n_l).$$

In order to add vehicles in the words, we must require $\mathscr{A}_1(n_k) \neq \emptyset$ or $\mathscr{A}_1(n_l) \neq \emptyset$, but this is not enough. It is necessary to define some compatible conditions over the pairs $(n_k, n_l) \in Q_5^2$. We say that (n_k, n_l) is compatible with class $c(i)$ for $i = 1, 2, 3$ if it satisfies the corresponding property defined below:

$c(1)$: $\mathscr{A}_1(n_k) \cap (I_A^+ \cup I_A^-) \neq \emptyset$ and $\mathscr{A}_1(n_l) \cap (I_B^+ \cup I_B^- \cup I_G) \neq \emptyset$,

$c(2)$: $\mathscr{A}_1(n_k) \cap (I_A^+ \cup I_A^-) \neq \emptyset$,

$c(3)$: $\mathscr{A}_1(n_l) \cap (I_B^+ \cup I_B^- \cup I_G) \neq \emptyset$.

These conditions ensure that the change of instruction in the preimage is done from the outside of the interval of cells between n_k and n_l. Then, the limit components which are added in each case are:

If $(n_k, n_l) \in c(1)$ then

$$\Lambda(\tilde{F})_{34}(n_k, n_l) = \{(R \vee 0)^* (V^{\mathrm{L}}(n_k) \vee \xi) 0^* n_k 0^* (V^{\mathrm{R}}(n_k) \vee \xi) 0^* (V^{\mathrm{L}}(n_l) \vee \xi) 0^* n_l 0^* (V^{\mathrm{R}}(n_l) \vee \xi)(L \vee 0)^*\},$$

$$\Lambda(\tilde{F})_{34} = \bigcup_{(n_k, n_l) \in c(1)} \Lambda(\tilde{F})_{34}(n_k, n_l):$$

if $(n_k, n_l) \in c(2)$ then

$$\Lambda(\tilde{F})_{35}(n_k, n_l) = \{(R \vee 0)^* (V^{\mathrm{L}}(n_k) \vee \xi) 0^* n_k 0^* (V^{\mathrm{R}}(n_k) \vee \xi) 0^* n_l (L \vee 0)^*\},$$

$$\Lambda(\tilde{F})_{35} = \bigcup_{(n_k, n_l) \in c(2)} \Lambda(\tilde{F})_{35}(n_k, n_l):$$

if $(n_k, n_l) \in c(3)$ then

$$\Lambda(\tilde{F})_{36}(n_k, n_l) = \{(R \vee 0)^* n_k 0^* (V^{\mathrm{L}}(n_l) \vee \xi) 0^* n_l 0^* (V^{\mathrm{R}}(n_l) \vee \xi)(L \vee 0)^*\},$$

$$\Lambda(\tilde{F})_{36} = \bigcup_{(n_k, n_l) \in c(3)} \Lambda(\tilde{F})_{36}(n_k, n_l).$$

(3.3) *A finite number of instructions.* Let $(n_k, n_{k_1}, \ldots, n_{k_p}, n_l)$ be a finite family of instructions in Q_5. As in (3.2), the unique components of the limit set are related to the class of compatibilities of the pair (n_k, n_l), and the general case without vehicles is

$$\Lambda(\tilde{F})_{37} = \{(R \vee 0)^* (n \vee 0)^* (L \vee 0)^*\}, \quad n = n_1 \vee n_2 \vee \cdots \vee n_{k_0}, \quad n_i \in Q_5, i = 1, \ldots, k_0.$$

If $(n_k, n_l) \in c(1)$ then

$$\Lambda(\tilde{F})_{38}(n_k, n_l) = \{(R \vee 0)^* (V^{\mathrm{L}}(n_k) \vee \xi) 0^* n_k 0^* (V^{\mathrm{R}}(n_k) \vee \xi)(n \vee 0)^* (V^{\mathrm{L}}(n_l) \vee \xi) 0^* n_l 0^* (V^{\mathrm{R}}(n_l) \vee \xi)(L \vee 0)^*\},$$

$$\Lambda(\tilde{F})_{38} = \bigcup_{(n_k, n_l) \in c(1)} \Lambda(\tilde{F})_{38}(n_k, n_l).$$

If $(n_k, n_l) \in c(2)$ then

$$\Lambda(\tilde{F})_{39}(n_k, n_l) = \{(R \vee 0)^*(V^{\mathrm{L}}(n_k) \vee \xi)0^* n_k 0^*(V^{\mathrm{R}}(n_k) \vee \xi) (n \vee 0)^* n_l (L \vee 0)^*\},$$

$$\Lambda(\tilde{F})_{39} = \bigcup_{(n_k, n_l) \in c(2)} \Lambda(\tilde{F})_{39}(n_k, n_l).$$

If $(n_k, n_l) \in c(3)$ then

$$\Lambda(\tilde{F})_{3,10}(n_k, n_l) = \{(R \vee 0)^* n_k (n \vee 0)^*(V^{\mathrm{L}}(n_l) \vee \xi)0^* n_l 0^* (V^{\mathrm{R}}(n_l) \vee \xi)(L \vee 0)^*\},$$

$$\Lambda(\tilde{F})_{3,10} = \bigcup_{(n_k, n_l) \in c(3)} \Lambda(\tilde{F})_{3,10}(n_k, n_l).$$

Then $\Lambda_3(\tilde{F}) = \bigcup_{j=1}^{10} \Lambda_{3j}(\tilde{F})$ produces a regular component because it is a finite union of regular components. Note that the case where the interaction occurs between instructions is covered by this point.

(4) *Interaction between markers and instructions.* It is not difficult to see that the only component associated in this case is

$$\Lambda_4(\tilde{F}) = \{0^* F_A^* (n \vee 0)^* F_B^* 0^*\},$$

where $n = n_1 \vee n_2 \vee \cdots \vee n_{k_0}$, $n_i \in Q_5$, $i = 1, \ldots, k_0$.

In (1)–(4) above, we have studied all possible nondynamical interactions between states of $\tilde{F}$. The complete component that results is $\Lambda^1 = \bigcup_{i=1}^{4} \Lambda_i(\tilde{F})$, which is a finite union of regular components. Then, Λ^1 is regular, and the result is proved. □

Note that the result established in Proposition 3.9 does not depend on the perturbation introduced in M_{p}.

Now the problem of regularity has been reduced to the study of the components of the limit language associated with the dynamical configurations. To begin the study of this kind of configurations, it is useful to introduce the following definitions over the graph associated with a programmable machine.

Definition 3.10. Let M_{p} be a programmable machine and $G(M_{\mathrm{p}}) = (\mathcal{N}(M_{\mathrm{p}}), \mathcal{E}(M_{\mathrm{p}}))$ be its graph.

(1) We say that the edge $(n_i, n_j) \in \mathcal{E}(M_{\mathrm{p}})$ is of type 1 if

$$p_i = Go(n) \Rightarrow j = n,$$

$$p_i \neq Go(n) \Rightarrow j = i + 1.$$

(2) We say that the edge $(n_i, n_j) \in \mathcal{E}(M_{\mathrm{p}})$ is of type 2 if

$$n_i = (i, R_k^-(j)) \quad \text{for some } k \in \{1, \ldots, N\}.$$

In general, we make explicit the register where the jump occurs; in this case we say that (n_i, n_j) is of type 2 in R_k.

This definition makes explicit the differences between the type of actions of the nodes. For instance, in the case $n_1=(1, A^-(3))$, $n_2=(2, A^+)$ and $n_3=(3, Go(1))$, the connections (n_1, n_2), (n_3, n_1) and (n_2, n_3) are of type 1 and (n_1, n_3) is of type 2.

Definition 3.11. Let M_p be a programmable machine and $G(M_p)$ be its graph. Consider the s-tuple of nodes $\mathscr{C}=(n_{i_1}, n_{i_2}, \ldots, n_{i_s})$.

(1) $\mathscr{C}$ is a path iff $(n_{i_k}, n_{i_{k+1}})\in\mathscr{E}(M_p)$, $k=1, \ldots, s-1$.

(2) $\mathscr{C}$ is a circuit iff $\mathscr{C}$ is a path and $(n_{i_s}, n_{i_1})\in\mathscr{E}(M_p)$.

(3) $\mathscr{C}$ is a path of type 1 iff $(n_{i_k}, n_{i_{k+1}})$ is of type 1, $k=1, \ldots, s-1$.

(4) $\mathscr{C}$ is a path of type 2 iff $\exists k\in\{1, \ldots, s-1\}$ such that $(n_{i_k}, n_{i_{k+1}})$ is of type 2.

In this case, we say that $\mathscr{C}$ is of type 2 in $(R_j)_{j\in J}$, $J\subset\{1, \ldots, N\}$, referring to the registers where the jumps are produced.

When the PM evolves from initial conditions in the registers, over a path $\mathscr{C}$, it is not obvious if the run can be made; so, for each path, there are some minimal conditions over the registers that need to be satisfied to permit the evolution. For instance, for the path $\mathscr{C}=((1, A^+), (2, A^-(5)), (5, A^+))$ it is clear that there is no way to make the jump n_2–n_5, independently of the content of the registers.

Definition 3.12. Let M_p be a programmable machine and $G(M_p)$ its graph. Consider the path $\mathscr{C}=(n_{i_1}, \ldots, n_{i_s})$.

(1) We say that $\mathscr{C}$ is legal iff $\exists(a_j)_{j=1}^N\subseteq\mathbb{N}$ such that if $|R_j|=a_j$ then $\mathscr{C}$ can be traversed from this condition.

(2) Let $\mathscr{C}$ be a legal path. The base of $\mathscr{C}$ is defined as

$$M(\mathscr{C})=(M_1(\mathscr{C}), \ldots, M_N(\mathscr{C})),$$

where $M_k(\mathscr{C})=\min\{a_k\in\mathbb{N}\,|\,a_k=\tilde{a}_k, (\tilde{a}_i)_{i=1}^N \text{ makes } \mathscr{C} \text{ legal}\}$.

(3) Let $\mathscr{C}$ be a legal path. The variation index associated with $\mathscr{C}$ is defined as

$$\sigma(\mathscr{C})=(\sigma_1(\mathscr{C}), \ldots, \sigma_N(\mathscr{C})),$$

where

$$\sigma_k(\mathscr{C})=\sum_{j=1}^{s} sig_k(n_{i_j}, \mathscr{C}), \quad k=1, \ldots, N,$$

$$sig_k(n_{i_j}, \mathscr{C})=\begin{cases} +1 & \text{if } p_{i_j}=R_k^+, \\ -1 & \text{if } p_{i_j}=R_k^-(n),\ n\neq n_{i_{j+1}}, \\ 0 & \text{otherwise.}\end{cases}$$

With this notation, the minimal result in the registers associated with a run over the path $\mathscr{C}$ is

$$m(\mathscr{C})=M(\mathscr{C})+\sigma(\mathscr{C}),$$

where "+" is the sum in $\mathbb{R}^N$.

For instance, take $\mathscr{C}=((1, A^+), (2, B^-(1)), (3, A^+), (4, Go(1)))$, for initial contents $|A_0|$ and $|B_0|$, respectively, at the registers. At the end of the travel along the path, the resulting contents will be $|A|=|A_0|+2$ and $|B|=|B_0|-1$; then

$$\sigma(\mathscr{C})=(2, -1),$$

$$M(\mathscr{C})=(0, 1),$$

$$m(\mathscr{C})=(2, 0).$$

Now, let us show the following proposition.

Proposition 3.13. *Let M_T be a Turing machine and $\tilde{F}$ the PS associated with it. Then, the limit component associated with dynamical configurations is regular.*

Proof. We have to prove that the set of words in the limit language that consider interactions between all types of states (i.e. markers, vehicles and instructions) constitutes a regular language.

First of all, we study the contents at each register that allow an infinite inverse path from an instruction $n_k \in (Q_5 \setminus I_H)$ in the program $\mathscr{P}$ of the programmable machine. Let us describe these contents in the following form:

$$F_A \quad L_A(n_k)\ n_k\ L_B(n_k)\ F_B,$$

where $L_A(n_k)$ and $L_B(n_k)$ are words in the one-symbol alphabet $\{0\}$.

We divide the study according to the connection type of the elements in $\mathscr{A}(n_k)$, for all $n_k \in (Q_5 \setminus I_H)$ (observe that $|Q_5| < +\infty$).

(1) *Connections of type 1.* Let $n_k \in (Q_5 \setminus I_H)$. It is easy to prove that one and only one of the following conditions holds:

(i) n_k belongs to a unique circuit of type 1.

(ii) n_k belongs to a unique path (which is not a circuit) of type 1, called transient.

To have an infinite preimage $a \in \mathscr{A}(n_k)$ of type 1, n_k must satisfy (i). The circuit $\mathscr{C}$ associated with n_k must have a negative variation, $\sigma(\mathscr{C})$, in order to make possible the inversion.

With these conditions, an element $a \in \mathscr{A}(n_k)$ of type 1 is valid for the inversion iff n_k belongs to a type-1 circuit $\mathscr{C}$ and $\sigma(\mathscr{C}) \leqslant 0$. But, for our simulation, we consider the PM constructed by IB and DB blocks, which have circuits with $\sigma(\mathscr{C}) \not\leqslant 0$, and nodes corresponding to the external loops of perturbations, which are associated with circuits with index $\sigma(\mathscr{C})=(-1, 0)$ or $(0, -1)$. In the last case, it is easy to see that the structure of contents at the registers is

$$F_A\ 0^*\ n_k\ 0^*\ F_B. \tag{3.4}$$

(2) *Connections of type 2.* Consider the instruction $n_k \in (Q_5 \setminus I_H)$ and $a \in \mathscr{A}(n_k)$ be of type 2, i.e. there exists at least one type-2 connection in a. Let $a_i=(j_i, A^-(j_{i-1}))$ (or $(j_i, B^-(j_{i-1}))$ be the first node in a with connection of type 2 and j_{i-1} the instruction associated with a_{i-1}. To make the transition (a_i, a_{i+1}), either register A (or B) must be

empty at a_i. With this condition, the future is determined in either register A (or B), but not both. This problem is solved using the perturbed nodes associated with a_i.

Consider the node a_i, and let us denote it by g_a^- if it is associated with A, and by g_b^- in the other case, when it is used like a jump. An element in $\mathscr{A}(a_i)$ is the following perturbation circuit or external loop when $a_i = g_a^-$ (the case $a_i = g_b^-$ is symmetric):

$$(a_i, \tilde{a}_i = A^-(a_i), Go(\tilde{a}_i), \tilde{a}_i, Go(\tilde{a}_i), \ldots)$$
$$\mathscr{C}$$

Since $\sigma(\mathscr{C}) \leqslant 0$, it is clear that $\mathscr{C}$ is a valid circuit of type 1 for the inversion. Moreover, it allows infinite preimages for content zero in A and for arbitrary size in B, at a_i (or the symmetric one if $a_i = g_b^-$).

Then, the preimages of type 2 for all the nodes are determined, starting from the conditional jumps g_a^- and g_b^-. The contents look like

$$F_A \; g_a^- \; 0^* \; F_B \quad \text{and} \quad F_A \; 0^* \; g_b^- \; F_B. \tag{3.5}$$

In the sequel we just consider the case g_a^-; the other case is symmetric. We will study the evolution of type 1 from expression (3.5). It is clear, from the structure of the blocks DB and IB, that expression (3.5) evolves to a circuit $\mathscr{C}$ of type 1, such that (see Fig. 5)

$$\sigma_A(\mathscr{C}) \geqslant 0 \quad \text{and} \quad \sigma_B(\mathscr{C}) \leqslant 0. \tag{3.6}$$

Let n be an instruction in the circuit $\mathscr{C}$ such that

$$a_i = g_a^- \to (n_{i_1}, n_{i_2}, \ldots, n_{i_p}, n, n_{i_{p+2}}, \ldots, n_{i_T}) = \mathscr{C},$$
$$a_i = g_a^- \to (n_{i_1}, \ldots, n_{i_p}) \to (n, n_{i_{p+2}}, \ldots, n_{i_T}, n_{i_1}, \ldots, n_{i_p});$$
$$\tau \qquad\qquad\qquad \mathscr{C}'$$

then

$$\sigma_A(\tau) \geqslant 0, \qquad \sigma_A(\mathscr{C}') \geqslant 0 \quad \text{and} \quad \sigma_B(\mathscr{C}') \leqslant 0. \tag{3.7}$$

Now we study the contents of the registers A and B, that permit the inverse evolution through circuit $\mathscr{C}'$ and transient τ from instruction n.

It is clear that, when we travel τ, the contents reached at instruction n must satisfy the equations

$$|A| = m_A(\tau), \qquad |B| \geqslant m_B(\tau). \tag{3.8}$$

$$\begin{array}{cccccc}
 & & F_A & a_i & 0^* & F_B \\
 & & F_A & \tilde{a}_i & 0^* & F_B \\
 & & F_A & Go(\tilde{a}_i) & 0^* & F_B \\
 & F_A & 0 & \tilde{a}_i & 0^* & F_B \\
 & F_A & 0 & Go(\tilde{a}_i) & 0^* & F_B \\
F_A & 0 & 0 & \tilde{a}_i & 0^* & F_B
\end{array}$$

Fig. 5. The preimages associated with the perturbation node $\tilde{a}_i$.

From equations (3.8), for each travel of the circuit $\mathscr{C}'$, the new content in register A at instruction n is

$$|A| = m_A(\tau) + k\sigma_A(\mathscr{C}'),$$

where k represents the kth iteration of $\mathscr{C}'$.

In register B, to obtain an inverse dynamics from the kth iteration of $\mathscr{C}'$ at instruction n, we have to verify the condition $|B| \geqslant m_B(\mathscr{C}')$. For this condition, it is necessary, at the beginning of the travel after the transient τ, to have

$$|B| \geqslant m_B(\mathscr{C}') - k\sigma_B(\mathscr{C}') \tag{3.9}$$

and in the jump instruction a_i it is verified that

$$|B| \geqslant m_B(\mathscr{C}') - k\sigma_B(\mathscr{C}') - \sigma_B(\tau). \tag{3.10}$$

On the other hand, the content of B which allows the travel of τ is $M_B(\tau)$; then

$$m_B(\mathscr{C}') - k\sigma_B(\mathscr{C}') - \sigma_B(\tau) \geqslant M_B(\tau). \tag{3.11}$$

So, it is necessary that

$$k \geqslant \frac{m_B(\tau) - m_B(\mathscr{C}')}{|\sigma_B(\mathscr{C}')|}. \tag{3.12}$$

Hence, there must exist $k_0 \in \mathbb{N}$, such that for any $k \geqslant k_0$ equation (3.11) holds.

For $k < k_0$, we consider the contents $f(k)$, at the kth travel of $\mathscr{C}'$ in n, that satisfies the following equality:

$$f(k) - k\sigma_B(\mathscr{C}') - \sigma_B(\tau) = M_B(\tau). \tag{3.13}$$

Then the contents generated, starting from equation (3.5), are

$$\begin{aligned} &F_A(0^{\sigma_A(\mathscr{C}')})^{k_A} 0^{m_A(\tau)} n 0^{f(k_A)} 0^* \quad \text{if } k_A < k_0, \\ &F_A(0^{\sigma_A(\mathscr{C}')})^{k_A} 0^{m_A(\tau)} n 0^{m_B(\tau)} 0^* \quad \text{if } k_A \geqslant k_0. \end{aligned} \tag{3.14}$$

Since k_A is independent of the content of B, for $k_A \geqslant k_0$ the general structure of contents that have inverse dynamics can be written as follows:

$$F_A(0^{T_A})^* 0^{K_A} n 0^{K_B} (0^{T_B})^* F_B. \tag{3.15}$$

It is important to note that there is a finite number of such structures associated with each $n \in (Q_5 \setminus I_H)$ and that the cases $k_A < k_0$ are considered with $T_A = 0$ and $T_B = 1$.

Finally, by inspection of equation (3.4), we can conclude that expression (3.15) is the general structure of contents at each $n \in (Q_5 \setminus I_H)$.

Now we have to construct the associated limit components.

From previous lemmas and the proof of Proposition 3.9, it is enough to study the following types of words:

$$(\xi \vee (R \vee 0)^* 0) F_A F_A^* w_1 n w_2 (L \vee 0)^*, \tag{3.16}$$

$$(R \vee 0)^* w_1 n w_2 F_B F_B^* (0(L \vee 0)^* \vee \xi), \tag{3.17}$$

$$(\xi \vee (R \vee 0)^* 0) F_A F_A^* w_1 n w_2 F_B F_B^* (0(L \vee 0)^* \vee \xi), \tag{3.18}$$

where $n \in Q_5$ and $w_i \in \{L, 0, R\}^*$, $i = 1, 2$.

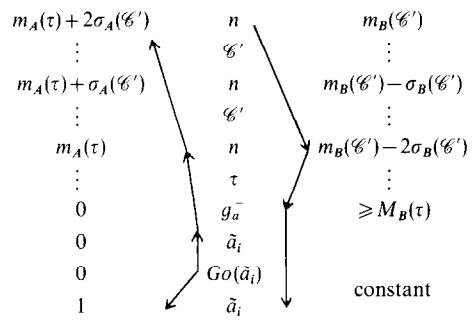


Fig. 6. Procedure of the proof of Proposition 3.13. We show the contents allowed at instruction $n\in(Q_5\setminus I_H)$, in differents steps from a g_a^-, and the preimages introduced with the loop.

When we consider the existence of vehicles, it is necessary to study the allowed contents in each register, that permit the inverse evolution of the programmable machine from the instruction n.

In the case of expression (3.16) (or (3.17)), it follows directly that the structure of contents in register A is the following one:

$$F_A(0^{T(n)})^*0^{k(n)}n,$$

where $k(n)$ and $T(n)$ are constants that depend on the instruction n.

The case of expression (3.18) corresponds to the preceding study. Note that these structures constitute regular languages (see expression (3.15)).

To finish, we have to transfer this regular structure to the limit language, by adding the corresponding vehicles. We sketch the procedure in an example.

Let $n_k=(k, A^+)$ and let $F_A(0^{T_A})^*0^{K_A}n_k0^{K_B}(0^{T_B})^*F_B$ be a structure of contents associated with the instruction $n_k\in Q_5$. Define the blocks

$$B_j^1=(0^{j-1}A^+0^{K_A-j}) \quad \forall j=1,\ldots,K_A+1,$$

$$B_j^2=(0^{j-1}A^+0^{T_A-j}) \quad \forall j=1,\ldots,T_A.$$

By using these blocks, the components of the limit language are described by the following expression:

$$F_A(0^{T_A})^*B_j^2(0^{T_A})^*0^{K_A}n_k0^{K_B+1}(0^{T_B})^*F_B,$$

$$F_A(0^{T_A})^*B_j^1n_k0^{K_B+1}(0^{T_B})^*F_B.$$

Similar blocks can be defined for the return states r and l.

This procedure is repeated a finite number of times for each $n_k\in Q_5$. Then, the component associated with dynamical configurations, Λ^2, is a regular language.

Proof of Theorem 3.4. In Propositions 3.9 and 3.13 we construct all the components of the limit language. Both are regulars, then, the limit language $\mathscr{L}_{\tilde{F}}=\Lambda^1\cup\Lambda^2$ is regular. □

Corollary 3.14. *Let U be a universal Turing machine. Then, there exists a linear cellular automaton $\tilde{F}$ which simulates it such that its limit language $\mathscr{L}_{\tilde{F}}$ is regular.*

This result implies that the computational complexity of a CA is not necessarily supported by the limit language complexity.

Finally, it is important to remark that Proposition 3.9 is true for the normal and perturbed simulator. Moreover, the study made in Proposition 3.13 shows that the limit language complexity depends only on the structure of contents at the registers that permit inverse dynamics in the programmable machine at each $n_k \in Q_5$. Then, for more general structures of programmable machines (not not only those composed by blocks IB and DB), we just have to study the connection graph of the programmable machine.

4. Non-perturbed programmable machine

From the proof of Theorem 3.4, it is clear that the limit language is regular when for each instruction g_a^- and g_b^- the following conditions over the contents of the registers are satisfied:

For a finite number of couples $(K, T) \in \mathbb{N}^2$,

$$F_A g_a^- 0^K (0^T)^* F_B \quad \text{have infinite preimages,} \tag{4.1}$$

$$F_A 0^K (0^T)^* g_b^- F_B \quad \text{have infinite preimages.} \tag{4.2}$$

An equivalent condition is that the languages L_A and L_B, representing the allowed contents in the registers at each jump instruction g_a^- and g_b^-, be regulars or, equivalently, context-free [10]. The perturbation introduced in Section 3 is an example where conditions (4.1) and (4.2) hold. There, $K=0$ and $T=1$.

In the case of programmable machines developed in Theorem 2.1, it is not difficult to see that for each g_a^- and g_b^- there exist programmable machines which have an uncountable family of languages of the type $0^K(0^T)^*$ that satisfy conditions (4.1) and (4.2). It is clear that the regularity depends on the finiteness of this family. In general, this finiteness is not obvious.

It is important to point out that the limit language complexity is the same as the complexities of languages L_A and L_B at registers A and B, associated with the inversion from instruction g_a^- and g_b^-.

Proposition 4.1. *Let F be the cellular automaton developed in Section 2. If $\mathscr{L}_F$ is regular, or context-free, or context-sensitive, or recursively enumerable, then L_A and L_B belong to the same class of language as $\mathscr{L}_F$.*

Proof. Follows directly from the fact that this class of languages is closed under intersection with a regular language. Consider the regular language $\mathscr{R} = \{F_A 0^* 0 g_b^- B^- F_B\}$;

then, $\mathscr{L}_F \cap R = \{F_A L_A 0 g_b^- B^- F_B\}$. The complexity of this language depends only on L_A. Similarly, we prove the property for a g_a^-. □

The proof of the converse of Proposition 4.1 is straightforward when L_A and L_B are regular or context-free languages. We just use the block system developed in Section 3.

Corollary 4.2. *Let F be the cellular automaton developed in Section 2; then*

$$\mathscr{L}_F \text{ is regular iff } \forall g_a^-, g_b^- \text{ the languages } L_A \text{ and } L_B \text{ are regulars.}$$

Now we give an example where this class is reached for a nonperturbed PM.

Proposition 4.3. *Let M_T be a deterministic finite automaton with a two-symbol alphabet and F the normal simulator associated with it. Then $\mathscr{L}_F$ is regular.*

Proof. For this particular class of Turing machines, it is not difficult to see that the blocks associated with each $q_i \in Q$ in the corresponding programmable machine are reduced to the blocks in Fig. 7.

We just have to study the connections (g_a^-, g_b^-) and (g_b^-, g_a^-), in the sense of conditions (4.1) and (4.2). The preimages associated with these instructions are the following:

$$\begin{array}{l} \dfrac{(0^2)^j \vee (0^2)^j 0}{0^k} \\ \dfrac{(0^2)^j \vee (0^2)^j 0}{0^k} \end{array} \quad \begin{array}{l} g_a^- \\ g_b^- \\ g_a^- \\ g_b^- \\ \vdots \end{array} \quad \begin{array}{l} 0^j \\ \\ \dfrac{0^k}{0^j} \\ \\ \\ (1) \end{array} \qquad \text{or} \qquad \begin{array}{l} 0^j \\ \dfrac{(0^2)^k \vee (0^2)^k 0}{0^j} \end{array} \quad \begin{array}{l} g_b^- \\ g_a^- \\ g_b^- \\ \vdots \\ \vdots \\ (2) \end{array} \quad \begin{array}{l} 0^j \\ \dfrac{}{0^k} \end{array}$$

From (1) and (2), it is clear that the contents allowed in each g_a^-, g_b^- are

$$0 | g_a^- 0^* \quad \text{and} \quad 0^* g_b^- | 0.$$

The languages L_A and L_B are the same for all the preimages. Then $\mathscr{L}_F$ is regular. □

In the case of a normal simulator for a universal Turing machine, we can use a result of undecidability to prove that the limit language is not recursively enumerable. In [8], it was proved that for some kind of simulations of Turing machines, Ψ_T, it is

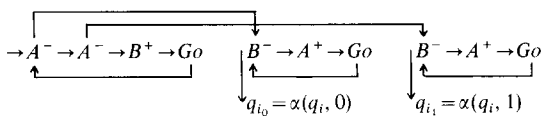


Fig. 7. Transition block $B(q_i)$ associated with the state q_i in a finite automaton. All blocks $B(q_i)$ are identical for any $q_i \in Q$.

undecidable whether $(q, s) \in (Q \times S)$, a lecture–state pair is in the limit language of Ψ_T. Equivalently, we can prove that it is, in general, undecidable whether the programmable machine has inverse dynamics from an instruction n_k and contents $|A|$ and $|B|$ in the registers.

Theorem 4.4. *Let U be a universal Turing machine and F_U the normal simulator associated with it; then $\mathscr{L}_{F_U}$ is not recursively enumerable.*

Proof. Direct from previous observation. □

The preceding examples show that the nonperturbed simulation preserves, in some sense, the computational complexity of the CA and, on the other hand, supports the regularity for simple Turing machines. More complex examples may be conjectured from characterizations of context-sensitive and recursively enumerable languages with one-symbol alphabet.

5. Limit complexity for an arbitrary PM

In previous sections we have just considered programmable machines which appear in the simulation of a Turing machine, in Theorem 2.1. All of them are interconnections of blocks IB and DB. Now we will consider any programmable machine with two registers.

The first result appears by combining Theorems 2.1 and 3.4.

Theorem 5.1. *Let M_p be a programmable machine. Then, there exists a CA, F, which simulates M_p such that its limit language $\mathscr{L}_F$ is regular.*

Proof. By Theorem 2.1, there exists a Turing machine M_T which simulates M_p (in some sense, that we will not specify), and for M_T there exists a programmable machine $\tilde{M}_p$, constructed with blocks IB and DB, which simulate it. Then, by Theorem 3.4, if we take the PS, F, associated with $\tilde{M}_p$, we get that $\mathscr{L}_F$ is regular. The theorem is proved. □

On the other hand, if we consider the direct simulation of any programmable machine with two registers, there exist examples where the contents at the registers that allow the inversion are synchronized, i.e. they produce context-free limit languages.

Theorem 5.2. *Let M_p be a programmable machine with two registers and F be the associated PS. Then $\mathscr{L}_F$ is at least a context-free language. Moreover, there exist programmable machines with nonregular context-free limit languages.*

Proof. From the discussion of the previous sections, we just have to study the circuits of type 1 reached after a jump condition g_a^- or g_b^-.

It is not difficult to see that the only case that breaks the regularity occurs when, after a g_a^- or a g_b^-, there exist circuits of type 1, $\mathscr{C}$, such that $\sigma_A(\mathscr{C})>0$ and $\sigma_B(\mathscr{C})>0$.

In all these cases, the allowed contents generated at any $n_k \in (Q_5 \setminus I_H)$ that belong to such circuits are the following:

$$F_A(0^{\sigma_A(\mathscr{C})})^i 0^{m_A(\tau)} n_k 0^{m_B(\tau)} (0^{\sigma_B(\mathscr{C})})^i 0^* F_B, \quad i \geqslant 0,$$

$$F_A 0^* (0^{\sigma_A(\mathscr{C})})^i 0^{m_A(\tau)} n_k 0^{m_B(\tau)} (0^{\sigma_B(\mathscr{C})})^i F_B, \quad i \geqslant 0.$$

Here τ is the transient of type 1, that joins the states g_a^- or g_b^- with the circuit $\mathscr{C}$.

The language generated by this structure is context-free. It is similar to languages of type $L=\{a^n b^m, m \geqslant n\}$.

Let us show an example where this class is reached. It is defined by the following program.

1. $B^-(7)$
2. $B^-(4)$
3. $Go(1)$
4. B^+
5. A^+
6. $Go(4)$
7. H

From $n_2=(2, B^-(4))$, the program reaches the circuit

$$g_b^- = n_2 \rightarrow \underbrace{B^+ \rightarrow A^+ \rightarrow Go}_{\mathscr{C}},$$

with $\sigma(\mathscr{C})>0$. □

This example allows one to recover one of the complexities attained in [8], for a general class of cellular automata.

To summarize, the limit-language complexity depends only on the variations of the circuits of type 1 associated with each program. This gives an algorithm allowing one to identify the limit complexity for any programmable machine with two registers. Then there exists a finite algorithm to identify the limit complexity of $\mathscr{L}_F$, where F is the associated PS.

In a similar way, we can obtain more complex limit languages by simulating any programmable machine with three registers and by using programs with circuits $\mathscr{C}$ of type 1 such that $\sigma(\mathscr{C})>0$.

Example 5.3. Let M_p be a programmable machine with three registers, defined by the program:

1. $A^-(5)$
2. $B^-(4)$

3. $Go(1)$
4. H
5. A^+
6. B^+
7. C^+
8. $Go(5)$

It is clear that the contents reached from $A^-(1)$ are dependent and constitute languages which are equivalent to $L=\{a^jb^jb^*c^jc^*, j\geqslant 1\}$. This last language is context-sensitive non-context-free.

This example allows one to recover all the classes given in [8].

6. Conclusion

The perturbation introduced in Section 3 shows that the CA can be modified in order to obtain a recursive limit language. Also, the perturbation preserves the direct dynamics of the CA over the simulating configurations. Therefore, the limit language can be regular, independently of the computational complexity.

We have shown that the study of limit languages complexity can be made through an analysis of some particular class of one-symbol languages. These languages appear naturally in the inversion condition at jump instructions. Their characterization could give some insight into the problem of founding new classes of limit languages.

Acknowledgment

We thank Professors F. Blanchard and J. Mazoyer for helpful discussions.

References

[1] N. Chomsky, On certain formal properties of grammars, *Inform. and Control* **2** (1959)137–167.
[2] N. Chomsky and G.A. Miller, Finite state languages, *Inform. and Control* **1** (1958) 91–112.
[3] K. Culik II, J. Pachl and S. Yu, On the limit sets of cellular automata, *SIAM J. Comput.* **18** (1989) 831–842.
[4] K. Culik II and S. Yu, Undecidability of C.A. classification schemes, *Complex Systems* **2** (1988) 177–190.
[5] S. Eilenberg, *Automata, Languages and Machines, Vol. A* (Academic Press, New York, 1974).
[6] J.E. Hopcroft and J.A. Ullman, *Formal Languages and their Relation to Automata* (Addison-Wesley, Reading, MA, 1969).
[7] J.E. Hopcroft and J.A. Ullman, *Introduction to Automata Theory Languages and Computation* (Addison-Wesley, Reading, MA, 1979).
[8] L. Hurd, Formal language characterizations of cellular automaton limit sets, *Complex Systems* **1** (1987) 69–80.
[9] L. Hurd, The application of formal language theory to the dynamical behaviour of cellular automata, Ph.D. Thesis, Princeton University, 1988.

[10] H.R. Lewis and C.H. Papadimitriou, *Elements of the Theory of Computation* (Prentice-Hall, Englewood Cliffs, NJ, 1981).
[11] K. Lindgren and M. Nordahl, Complexity measures and cellular automata, *Complex Systems* **2** (1988) 409–440.
[12] A. Maass, Complejidad Límite para una máquina de Turing, Memoria de Ingeniero Civil Matemático, Depto. Ing. Mat., U. de Chile, 1990.
[13] A. Maass, Universal cellular automaton simulating any programmable machine, *Revista de Matemáticas Aplicadas* **12** (1991) 107–126.
[14] J. Mazoyer, A six-state minimal time solution to the firing squad synchronization problem, *Theoret. Comput. Sci.* **50** (1987) 183–238.
[15] M. Minsky, *Computation: Finite and Infinite Machines* (Prentice-Hall, Englewood Cliffs, NJ, 1967).
[16] B. Weiss, Subshifts of finite type and sofic systems, *Monatsh. Math.* **77** (1973) 462–474.
[17] S. Wolfram, Twenty problems in the theory of cellular automata, *Phys. Scripta* **9** (1984) 170–172.
[18] S. Wolfram, Computation theory of cellular automata, *Comm. Math. Phys.* **96** (1984) 15–57.